Spitfire ’91, a WPF Shmup

Victor Gasior and Jaxx ’Lucky’ Woods

Abstract

Spitfire ’91 is a retro inspired Shoot ’em Up game (or shmup) where you control a small airplane fighting drones, tanks, and enemy aircraft. The target audience for this project is fans of retro schups like EDF or UN Squadron. So far we have constructed just the basic plans for the project.

1. Introduction

Spitfire ’91 should be a fun, fast paced, retro inspired Shoot ’em Up game made in WPF / C Sharp. In Spitfire ’91 the player controls a modern military fighter jet and fights enemy jets, drones, and ground vehicles. The game is supposed to emulate an SNES (Super Nintendo Entertainment System) era Shoot ’em Up game with its own art, design, sounds, and mechanics. There will be at least one stage where the players can fight enemies in order to try to get a higher and higher high scores. The target audience for this game will be Shoot ’em Up enthusiasts and fans of retro styled video games in general. The audience should first and foremost enjoy the game, while also tailoring the experience to resemble a retro shmup without making it difficult for fans of modern video games to enjoy. There are not many Shoot ’em Up games produced any more so we are appealing to a niche audience starved of content.

1.1. Background

Shoot em’ Ups rely on a player controller a vehicle that can move around the screen and shoot enemies that are trying to attack the player. The player obtains score by avoiding and defeating enemies. The game ends once they reach the end of the level / stage or loses do to being hit by enemies too many times

The reason we decided to undertake this project is that Shoot ’em Ups are both fun to play, and seem interesting to design.

1.1.1. Important Terminology. Shump - Common abbreviation for Shoot ’em Up High Score - A goal of a Shoot ’em Up game, players try to earn higher scores.

1.2. Impacts

This game should bring enjoyment to the audience and provide a free modern alternative to old retro Shoot ’em Up games that can be hard to play due to old hardware and games not being ported to modern systems.

1.3. Challenges

Handling enemy, projectile, and player collision in WPF. Displaying information and moving backgrounds in WPF. Storing enemy and level data in WPF.

2. Scope

This game should the bare minimum contain one playable level with multiple enemy variations that can spawn. There should be a win and lose state, as well as a connected ’Game Over’ screen on loss. There should be a main menu screen the user accesses first before starting the main game. There should be a high score tracker which displays and saves the highest scores to add replay value. Current Stretch Goals are as follows: multiple levels with different backgrounds, a working, upgrade shop and currency system, game saving, and music

2.1. Requirements

As part of fleshing out the scope of your requirements, you’ll also need to keep in mind both your functional and non-functional requirements. These should be listed, and explained in detail as necessary. Use this area to explain how you gathered these requirements.

Use Case ID	Use Case Name	Primary Actor	Complexity	Priority
1	Add item to cart	Shopper	Med	1
2	Checkout	Shopper	Med	1

TABLE 1. SAMPLE USE CASE TABLE

2.1.1. Functional.

- User needs to have a private shopping cart – this cannot be shared between users, and needs to maintain state across subsequent visits to the site
- Users need to have website accounts – this will help track recent purchases, keep shopping cart records, etc.
- You'll need more than 2 of these...

2.1.2. Non-Functional.

- Security – user credentials must be encrypted on disk, users should be able to reset their passwords if forgotten
- you'll typically have fewer non-functional than functional requirements

2.2. Use Cases

This subsection is arguably part of how you define your project scope (why it is in the Scope section...). In a traditional Waterfall approach, as part of your requirements gathering phase (what does the product actually *need* to do?), you will typically sit down with a user to develop use cases.

You should have a table listing all use cases discussed in the document, the ID is just the order it is listed in, the name should be indicative of what should happen, the primary actor is typically most important in an application where you may have different levels of users (think admin vs normal user), complexity is a best-guess on your part as to how hard it should be. A lower number in priority indicates that it needs to happen sooner rather than later. A sample table, or Use Case Index can be seen in Table 1.

Use Case Number: 1
Use Case Name: Add item to cart
Description: A shopper on our site has identified an item they wish to buy. They will click on a "Add to Cart" button. This will kick off a process to add one instance of the item to their cart.

You will then go on to (minimally) discuss a basic flow for the process:

1) User navigates to page listing desired item
2) User left-clicks on "Add to Cart" button.
3) User cart is updated to reflect the new item, this also updates the current total.

Termination Outcome: The user now has a single instance of the item in their cart.

You may need to also add in any alternative flows:
Alternative: Item already exists in the cart

1) User navigates to page listing desired item
2) User left-clicks on "Add to Cart" button.
3) User cart is updated to reflect the new item, showing that one more instance of the existing item has been added. This also updates the current total.

Termination Outcome: The user now has multiple instances of the item in their cart.

You will often also need to include pictures or diagrams. It is quite common to see use-case diagrams in such write-ups. To properly reference an image, you will need to use the `figure` environment and will need to reference it in your text (via the `ref` command) (see Figure 1). NOTE: this is not a use case diagram, but a kitten.

After fully describing a use case, it is time to move on to the next use case:

Use Case Number: 2
Use Case Name: Checkout
Description: A shopper on our site has finished shopping. They will click on a "Checkout" button. This will kick off a process to calculate cart total, any taxes, shipping rates, and collect payment from the shopper.

You will then need to continue to flesh out all use cases you have identified for your project.

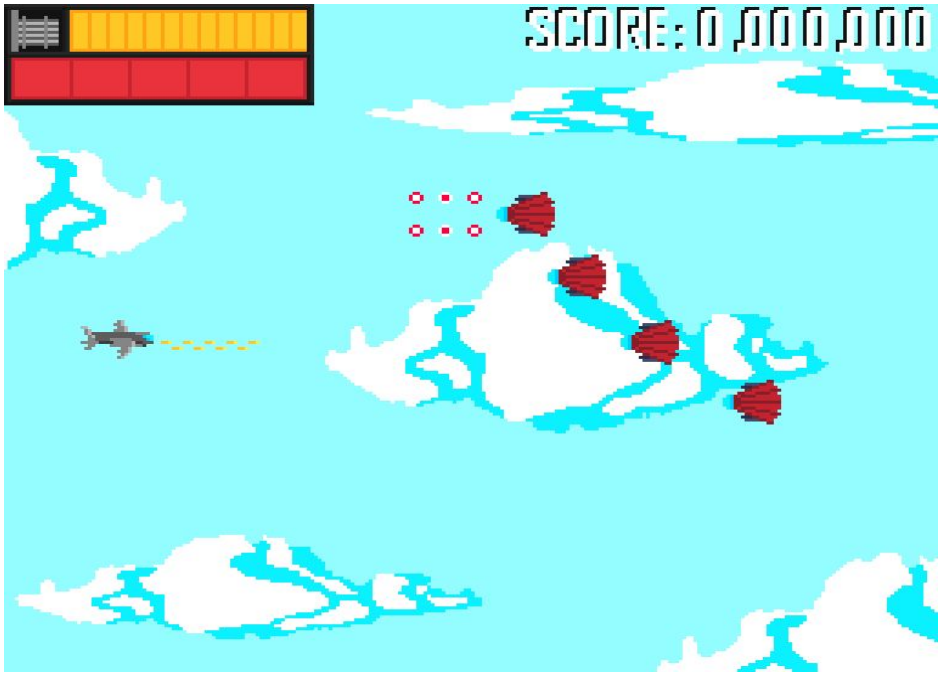

Figure 1. First picture, this is a kitten, not a use case diagram

2.3. Interface Mockups

At first, this will largely be completely made up, as you get further along in your project, and closer to a final product, this will typically become simple screenshots of your running application.

In this subsection, you will be showing what the screen should look like as the user moves through various use cases (make sure to tie the interface mockups back to the specific use cases they illustrate).

3. Project Timeline

Go back to your notes and look up a typical project development life cycle for the Waterfall approach. How will you follow this life cycle over the remainder of this semester? This will usually involve a chart showing your proposed timeline, with specific milestones plotted out. Make sure you have deliverable dates from the course schedule listed, with a plan to meet them (NOTE: these are generally optimistic deadlines).

4. Project Structure

At first, this will be a little empty (it will need to be filled in by the time you turn in your final report). This is your chance to discuss all of your design decisions (consider this the README's big brother).

4.1. UML Outline

Show the full structure of your program. Make sure to keep on updating this section as your project evolves (you often start out with one plan, but end up modifying things as you move along). As a note, while Dia fails miserably at generating pdfs (probably my fault), I have had much success with png files. Make sure to wrap your images in a `figure` environment, and to reference with the `ref` command. For example, see Figure 2.

4.2. Design Patterns Used

Make sure to actually use at least 2 design patterns from this class. This is not normally part of such documentation, but largely just specific to this class – I want to see you use the patterns!

5. Results

This section will start out a little vague, but it should grow as your project evolves. With each deliverable you hand in, give me a final summary of where your project stands. By the end, this should be a reflective section discussing how many of your original goals you managed to attain/how many desired use cases you implemented/how many extra features you added.

5.1. Future Work

Where are you going next with your project? For early deliverables, what are your next steps? (HINT: you will typically want to look back at your timeline and evaluate: did you meet your expected goals? Are you ahead of schedule? Did you decide to shift gears and implement a new feature?) By the end, what do you plan on doing with this project? Will you try to sell it? Set it on fire? Link to it on your resume and forget it exists?

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.